International Journal of Computer Science Engineering
and Information Technology Research (IJCSEITR)
ISSN(P): 2249-6831; ISSN(E): 2249-7943
Vol. 4, Issue 3, Jun 2014, 47-54

A STUDY ON ORPHAN PROCESSES IN THE DISTRIBUTED SYSTEMS ENVIRONMENTS

LIMI KALITA[1] & LAKSHMI PRASAD SAIKIA[2]

[1]M. Tech Student, Department of Computer Science & Engineering, Assam Down Town University, Guwahati, India

[2]Professor & HOD, Department of Computer Science & Engineering, Assam Down Town University, Guwahati, India

ABSTRACT

In distributed computing, orphan process are unwanted computations that creates problems like corruption of data, wastage of resources like CPU, memory, databases, squandering of server execution time and may lead to deadlock thus slowing down the performance of computers and affecting reliability of the entire system. This paper presents an overview on various issues in orphan handling and describes several techniques for detecting and terminating of the orphans in distributed systems.

KEYWORDS Orphan Process, CPU, Memory, Databases, Squandering, Deadlock

INTRODUCTION

Distributed computing is a field of computer science that studies distributed systems. A distributed system is composed of nodes/computers that are connected to any network and communicate with each other by passing messages. For e.g.: ATMs as shown in figure 1.

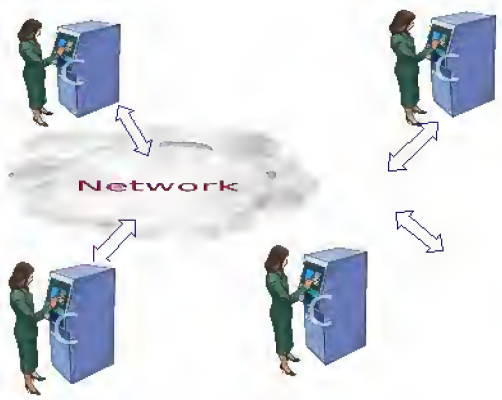

Figure 1: ATMs as a Distributed System

In this computing, different sections of a program are running simultaneously on two or more computers in the entire system. In multitasking computer systems, where remote procedure calls between clients and servers are intended, a parent process can create many child processes by making a system call fork() to the operating system's kernel as seen in figure 2(a). But if the parent process dies, then that child process is transformed into orphan process and it keeps running as shown in figure 2(b). An orphan process can be created unintentionally such as when the parent process crashes or terminates. Also orphans can be created intentionally so that it becomes detached from the user's session and left running in the background terminals.

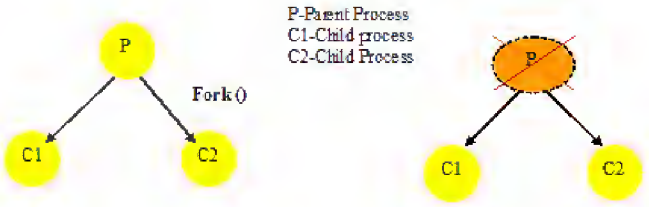

Figure 2(a): Process **Figure 2(b): Orphan Process**

So, orphan processes are unwanted activities executing on behalf of aborted and crashed processes and they need to be dealt with. They are undesirable because they waste system resources, wastage of CPU time, memory, databases, creates deadlock by holding locks/resources needed by non-orphans, produces inconsistent data etc. So here in this paper, various issues and approaches for detection and termination of these Orphans have been put forward which is found in the literature.

RELATED WORKS

In a distributed system, the client communicates with the server by implementing remote procedure calls. But what happens if a client sends a request to do some work and crashes before the server replies? At this point an orphan computation is created which is active and no client is waiting for the result.

Nelson [1981] proposed four approaches for termination of orphans. They are Extermination, Reincarnation, Gentle reincarnation, Expiration. In extermination, before a client sends a request to the server, the RPCs are recorded in a nonvolatile storage/disk so that after rebooting in case of a crash, the log/record is checked and the orphan is explicitly killed. But its disadvantage is that recovery from a crash is delayed until all orphans created by crash are tracked down and destroyed. In expiration, a deadline is assigned with every process. As soon as the deadline expires, the processes are terminated automatically. Its disadvantage is that non-orphans are also killed. In all the above methods, the orphan is detected and killed only after rebooting of the crash or abort client.

Lampson [1981] modified the Nelson's approach by introducing dead lining where querying is done up the parent processes to ascertain its dead and if the child process is an orphan. If the process is not an orphan its time limit is extended.

Allicin [1983] proposed an algorithm for detection of orphans which is divided into two halves-an abort–orphan detection and crash–orphan detection.

- **Abort Orphan:** The parent process is aborted, but its child processes are active.
- **Crash Orphan**: The parent process is crashed, but its child processes are active.

S. K. Shrivastava [1983] says that node crashes and message duplication are the undesirable events that give rise to orphans in distributed systems.

A reliable remote procedure (RPC) call mechanism is designed so that the client processes can interact with the server processes running on individual nodes of the distributed system and also try to abstract the abnormal situations like node crashes and lost messages. Consider a scenario where a client sends a message to the server provided a message handling facility where messages occasionally get lost. Essentially, in this scheme, a client's remote call is transformed into an appropriate message to the named server which performs the requested work and sends the result back to the client

and terminating of the call. But due to interruption in connection, the receiving server may behave as if it has not sent a message when it already has. Then the sender will resend the message again resulting in duplicate messages at the server. So in this way the previously sent messages become orphans which are unwanted by the client.

Again, Orphans are created because the client crashes just after sending a message (m1) to the server as depicted in the figure 3. After recovery of the client, it resends the message (m2) and depending upon which message has been accepted either work1or work2 will become Orphan. But one exception of this situation is that orphans do not produce any serious problems unless both messages (m1=m2) are same. (I. e. idempotency property).

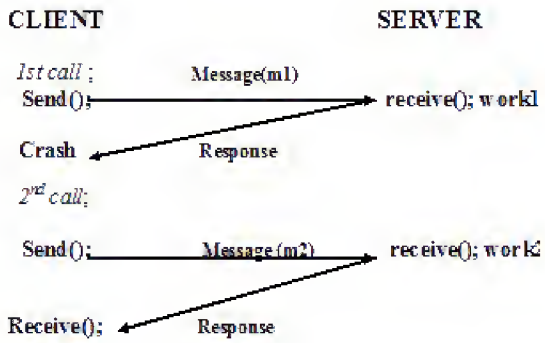

Figure 3: Work1 as Crash Orphan

'Exactly once' is a technique where RPCs are executed exactly once and only once or precisely if a client's call succeeds (i.e. the call does not return abnormally) then this implies that exactly one execution has taken place at the server 'At least once' is a technique where RPCs are executed at least one time but possibly more or precisely if a client call succeeds, then this implies that at least one execution has taken place at the server. Furthermore, classification is also possible (first one or last one) indicating which execution is responsible for termination of the call. A graph model of computation is utilized for discussing the sequence of events carried in the process by making use of Remote procedure calls. A directed acyclic graph has been to represent vertices and arcs as events and precedence relations between events respectively, which are called event graphs in the network.

MartinS. Mc Kendry and Maurice Heriihy [1985] proposed a new method for managing orphans created by crashes and by aborts. The method prevents orphans from observing inconsistent states, and ensures that orphans are detected and eliminated in a timely manner. A major advantage of this method is simplicity: it is easy to understand, to implement, and to prove correct. The method is based on timeouts using clocks local to each site. The entire approach revolves around the two approaches refresh protocol & termination protocol.

Refresh Protocol: A transaction that is not an orphan will be aborted unnecessarily if its quiesce time arrives at a site before its activity there completes. T o avoid this difficulty, a refresh protocol is periodically undertaken to advance each transaction's quiesce and release times. The interval between a site's current time and the quiesce time for any transaction is the quiesce interval, and the interval between the quiesce and release times is the release interval. The interval between refresh protocols is the refresh interval. The refresh protocol is a two-phase protocol. In the first phase, the home site attempts to advance the transaction's release time at all sites it has visited. If the first phase is successful, the home site attempts to advance the transaction's quiesce time at all sites visited. The two phases are necessary to ensure that the times are adjusted without violating the termination invariant. If phase1 completes successfully, the transaction's release time has been advanced at all sites. In phase2, the quiesce time is advanced.

The home site transmits a phase 2 refresh message advising visited sites of the new quiesce time. The termination invariant is preserved at each point during the protocol. Although responses to the phase 2 messages are not needed for correctness, they can reduce the likelihood of aborts caused by lost messages.

The Termination Protocol: When a transaction commits or aborts, its locks cannot be released until its release time has passed. To avoid waiting for a transaction's release time to arrive, a termination protocol can be used to adjust the release time without violating the termination invariant. The termination protocol is similar to the refresh protocol. The first phase attempts to move the quiesce time back to the present. When a transaction commits, the termination protocol can be integrated with the commit protocol. When a transaction aborts, an explicit abort protocol can be used to release its locks, or if the quiesce interval is acceptably small, its locks will gradually be released as its release times elapse.

Lakshmi Prasad Saikia and Kattamanchi Hemachandran [2007] have simulated a distributed system and carried out diagnosis under Arbitrary Network topologies in order to identify faulty nodes and fault-free nodes in the system by using Adaptive Distributed system-level diagnosis (ADSD) algorithm. In this algorithm, nodes test, or monitor, one another periodically such that each fault-free node is tested by exactly one other node. When a node detects that a fault –free node it is monitoring has become faulty or that a faulty neighbor has been repaired, it propagates this new information to its fault- free neighbors, which propagate the information to their neighbors and so on.

A distributed system consists of either fault-free or faulty nodes as shown in figure 4. A fault-free node is able to respond to a request or test within a specified deadline from its neighboring nodes in the system whereas a faulty node is unable to respond to a request from its neighboring nodes within a specified time-out period.

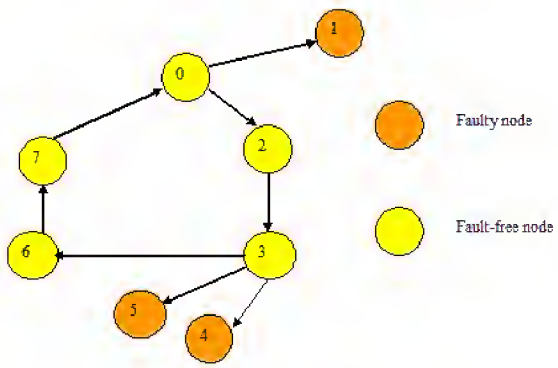

Figure 4: Example of Test Assignment in Adaptive –DSD [17]

Shamsdueen E and Dr. V. Sundaram [2011] proposed a global log approach, where orphans are killed immediately after their birth. In a distributed system, RPC is the interprocess communication mechanism for passing messages from client to server. So in this protocol, the system consists of a server process which sends a token to the clients in order to identify the active and passive clients in the entire system. The token contains the details of process id and status variables assigned to all client processes, i.e. '0', if the client has crashed or aborted and '1', if the client is alive. The token passes from one client to another in the system. After revolving round the token returns back to the monitor (server) process. Then the monitor process after detecting the client being assigned with status variable '0' will send a message to kill those client processes where orphan computations are active in the network. But due to any network failure or node failure, the token is not able to return back to the monitor. So in this method, a time-out period has been assigned

with the token before sending it to the clients in the network. So, killing of orphans takes place immediately after the node crash or abort process compared to the previous approaches. So the advantages of this protocol are no inconsistency of data and wastage of resources.

Shamsudeen. E and V. Sundaram [2013] deals with orphan process created by failure of nodes and abort process as shown in figure 5(a), figure 5(b) and figure 5(c). So the orphan processes continue to run but its results are no longer wanted because the parent processes that have initiated these orphans are either crashed or aborted. The orphans are unwanted computations causing problems like inconsistency of data, wastage of resources, wastage of server execution time, deadlock. A case study has also been made considering ATM performing banking transactions. Consider a scenario where we are visiting an ATM to withdraw money .So we insert the card and enter the PIN, then press Enter followed by the selection of transaction etc. But due to some communication failures or any technical/mechanical/power failures, the ATM fails to dispense the money. This situation has been created due to orphans. And the worst part of this situation is that the intended cash has been deducted from the account without dispensing the intended withdrawal at the user end. So orphan process should be detected and killed followed by their birth.

Figure 5 (a): The Client Process Sends a Request to Server Process

Figure 5 (b): While Server Process Computes the Result the Client Process Crash or Abort of Parent Process Happens

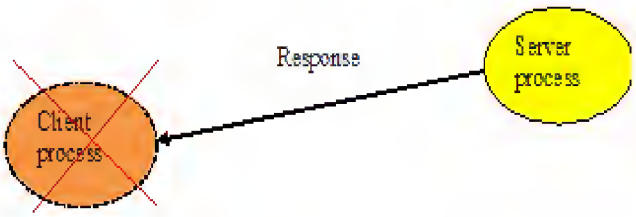

Figure 5(c): After Computation, the Server Sends Back the Result, but No Client is to Receive the Result

CONCLUSIONS

It is concluded that orphans are the undesirable events in a distributed system. So in this paper, various approaches had been put forward, their advantages and disadvantages in dealing with orphan processes. Orphan processes can make disastrous problems in our day to day life like in case of ATM. So, orphan processes are the villains whose results are unwanted and are to be detected and killed immediately after their birth.

REFERENCES

1. Eswaran, K.P., Gray, J.N., Lorie, R.A., and Traiger, I. L., "*The Notion of Consistency and Predicate Locks in a Database System*", Communications ACM 19(11): 624-633, November, 1976.
2. Lamport L. "*Time, clocks and the ordering of events in a distributed system*", Communications of the ACM 21(7):558-565, July, 1978.
3. Gray, J., "*Notes on Database Operating Systems*", Lecture Notes in Computer Science 60, Springer–Verlag, Berlin, pages 393-481, 1978.
4. B.J Nelson, "*Remote procedure call*" Ph.D. dissertation, Computer Science., Carnegie-Mellon Univ., Pittsburgh, P.A. Rep.CMU-CS-81-119,1981.
5. B. Lampson, "*Remote Procedure Calls"*, Lecture Notes in Computer Science 105. Springer-Verlag, Berlin, 365-370, 1981.
6. Santosh Shrivastava and Fabio Panzieri," *The design of a reliable Remote procedure call mechanism*", 1982.
7. S. K. Shrivastava "*On Treatment of Orphans in a Distributed Systems*", in Proc. 3rd Symp. Reliability in Distributed Software and Database Systems, IEEE Comput. Soc., Florida, Oct. 1983, pp. 155-162.
8. Allicin J.E, "*An architecture of reliable decentralized systems"*, Ph. D Thesis, Georgia Institute of Technology, September, 1983.
9. A.D. Birrell and B.J. Nelson, "*Implementing remote procedure calls*", ACM Trans. on Computer System, vol.2. no. 1, pp. 39-59, Feb 1984.
10. Martin S. McKendry Maurice Heriihy," *Time-Driven Orphan Elimination*" Computer Science Department Carnegie- Mellon University Pittsburgh, PA1521 3 1, July 1985.
11. Fabio Panzieri, Santosh K. Shrivastava," A *remote procedure call mechanism Supporting Orphan Detection and killing*" Proc. IEEE Transaction on Software Engineering, Vol. 14, No. 1, 1988.
12. Maurice Herlihy and Nancy Lynch, Michael Merritt and William Weihl, "*On the Correctness of Orphan Management Algorithms*", Journal of the Association for Computing Machinery, Vol. 39, No. 4, pp. 881-930, October 1992.
13. Valerie Issarny, Gilles Muller and Isabelle Puaut," *Efficient Treatment of failures in RPC systems",* Proc.13th Symposium on reliable Distributed systems, pp. 170-180. IEEE Comp. Society Press, 1994.
14. Jaochim Baumann and Kurt Rothermel," *The Shadow Approach: An Orphan Detection Protocol for Mobile Agents*", Springer-Verlag London Ltd, Personal Technologies 1998.
15. Per Brinch Hansen, "*Classic Operating System: From batch processing to Distributed systems*", Springer, 10-Jan-2001.
16. S. Pleisch, A. Kupsys, and A. Schiper," *Preventing Orphan Requests in the context of Replicated Invocation*", In Proc. 22nd Symp. On Reliable Distributed Systems, pages 119-129, 2003.

17. Lakshmi Prasad Saikia and Kattamanchi Hemachandan, "*Simulation of System Level Diagnosis in Distributed Arbitrary Network",* Journal of Theoritical and Applied Information Technology, 2007.

18. Pradeep K. Sinha," *Distributed Operating Systems Concepts and design*", Prentice hall India, 2008.

19. Shamsudeen. E and Dr. V. Sundaram, "*Time Stamp Based Global log and monitor approach to handle orphans in distributed systems"*, International journal of computer science and network security, vol 11 No.8, pp 123-125, 2011.

20. Ramez Elmasri, Shamkant D Navathe, "*Fundamentals of Database Management Systems*", 5th Edition, Pearson Educations, 6th Edition, 2011.

21. Shamsudeen E and V. Sundaram, "*Issues with orphan computations in distributed computing systems"*, International Journal of Computer Applications, pp.7-9, vol. 62, No. 20, Published by Foundation of Computer Science, Jan 2013.